# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

**4,800**
Open access books available

**122,000**
International authors and editors

**135M**
Downloads

Our authors are among the

**154**
Countries delivered to

**TOP 1%**
most cited scientists

**12.2%**
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

## Interested in publishing with us?
## Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

# 3

# A Multilevel Greedy Algorithm for the Satisfiability Problem

Noureddine Bouhmala[1] and Xing Cai[2]
[1]*Vestfold University College,*
[2]*Simula Research Laboratory,*
*Norway*

## 1. Introduction

The *satisfiability* (SAT) problem is known to be NP-complete [3] and plays a central role in many domains such as computer-aided design, computing theory, and artificial intelligence. Generally, a SAT problem is defined as follows. Given a propositional formula $\Phi = \bigwedge_{j=1}^{m} C_j$ with $m$ clauses and $n$ boolean variables, where each variable has value of either ***True*** or ***False***. Negation of boolean variable $x_i$ is denoted by $\bar{x}_i$. Each clause $C_j$ has the following form:

$$C_j = \bigvee_{k \in \mathcal{I}_j} \vee \bigvee_{k \in \bar{\mathcal{I}}_j},$$

where $\mathcal{I}_j$, $\bar{\mathcal{I}}_j$ are two sets of literals. The literals in $\mathcal{I}_j$ are from a subset of the $n$ boolean variables, and the literals in $\bar{\mathcal{I}}_j$ are from a subset of the negation of the $n$ boolean variables. Moreover, we have $\mathcal{I} \cap \bar{\mathcal{I}}_j = 0$. The task is to determine whether $\Phi$ evaluates to true. Such an assignment of the $n$ boolean variables, if it exists, is called a satisfying assignment for $\Phi$ (and $\Phi$ is called satisfiable). Otherwise $\Phi$ is said to be unsatisfiable. Most SAT solvers use a *conjunctive normal form* (CNF) representation of the formula $\Phi$. In CNF, the formula $\Phi$ is represented as a conjunction of clauses, each clause is a disjunction of literals, where a literal is a boolean variable or its negation. For example, $P \vee Q$ is a clause containing the two literals $P$ and $Q$. The clause $P \vee Q$ is satisfied if either $P$ is true or $Q$ is true. When each clause in $\Phi$ contains exactly $k$ literals, the restricted SAT problem is called $k$-SAT. In the numerical experiments of this chapter, we will focus on the 3-SAT problem, where each clause contains exactly 3 literals. Since we have two choices for each boolean variable, and taken over $n$ variables, the size of the search space $S$ is $|S| = 2^n$. The chapter is organized as follows. In Section 2 we review various algorithms for SAT problems. Section 3 explains the basic greedy GSAT algorithm. In Section 4, the multilevel paradigm is described. Section 5 presents the multilevel greedy algorithm. In Section 6, we look at the results from testing the new approach on a test suit of problem instances. Finally in Section 7 we give a summary of the work.

Source: Advances in Greedy Algorithms, Book edited by: Witold Bednorz,
ISBN 978-953-7619-27-5, pp. 586, November 2008, I-Tech, Vienna, Austria

## 2. Methods for SAT

The SAT problem has been extensively studied due to its simplicity and applicability. The simplicity of the problem coupled with its intractability makes it an ideal platform for exploring new algorithmic techniques. This has led to the development of many algorithms for solving SAT problems which usually fall into two main categories: systematic algorithms and local search algorithms. Systematic search algorithms are guaranteed to return a solution to a SAT problem if at least one exists or prove it insoluble otherwise.

### 2.1 Systematic search algorithms

The most popular and efficient systematic search algorithms for SAT are based on the Davis-Putnam (DP) [4] procedure which enumerates all possible variable assignments. This is achieved by setting up a binary search tree and proceeding until it either finds a satisfying truth assignment or concludes that no such assignment exists. In each recursive call of the algorithm the propositional formula $\Phi$ is simplified by unit propagation. A boolean variable $x_i$ is selected according to a predefined rule among the $n$ boolean variables. Next, find all the clauses that include the literal $x_i$ and delete it from all these clauses. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be the set of $k$(<= $m$) clauses resulting from this process. Similarly, let $\mathcal{D} = \{D_1, D_2, \ldots, D_r\}$ denote the set of $r$(<= $m$) clauses resulting from deleting the literal $\bar{x}_i$. Moreover, let $\mathcal{R} = \{R_1, R_2, \ldots, R_{(m-k-r)}\}$ represent the set of $m - k - r$ clauses that include neither of these two literals. Finally, the original propositional formula is reduced to

$$\Phi_{\text{simpler}} = \left( \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{r} (C_i \vee D_j) \right) \bigwedge_{l=1}^{m-r-k} R_l.$$

Note that the propositional formula $\Phi_{\text{simpler}}$ does not contain the boolean variable $x_i$ since none of the clauses set $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{R}$ include $x_i$. If thus an empty clause is obtained, the current partial assignment can not be extended to a satisfying one and backtracking is used to proceed with the search; if an empty formula is obtained, i.e., all clauses are satisfied, the algorithm returns a satisfying assignment. If neither of these two situations occur, an unassigned variable is chosen and the algorithm is called recursively after adding a unit clause containing this variable and its negation. If all branches are explored and no satisfying assignment has been reached, the formula is found to be unsatisfiable. For efficiency reasons, the search tree is explored in depth-first search manner. Since we are only interested in whether the SAT problem is satisfiable or not, we stop as soon as the first solution is found. The size of the search tree depends on the branching rule adopted (how to select the branch variable) thereby affecting the overall efficiency of DP. This has led to the development of various improved DP variants which differ in the schemes employed to maximize the efficiency of unit propagation in their branching rules.

### 2.2 Stochastic local search algorithms

Due to their combinatorial explosion nature, large and complex SAT problems are hard to solve using systematic algorithms. One way to overcome the combinatorial explosion is to give up completeness. Stochastic local search (SLS) algorithms are techniques which use this strategy. SLS algorithms are based on what is perhaps the oldest optimization method: *trial*

*and error*. Typically, they start with an initial assignment of the variables, either randomly or heuristically generated. Satisfiability can be formulated as an optimization problem in which the goal is to minimize the number of unsatisfied clauses. Thus, the optimum is obtained when the value of the objective function equals zero, which means that all clauses are satisfied. During each iteration, a new solution is selected from the neighborhood of the current one by performing a move. Choosing a good neighborhood and a search method are usually guided by intuition, because very little theory is available as a guide. Most SLS algorithms use a 1-flip neighborhood relation for which two truth value assignments are neighbors if they differ in the truth value of one variable. If the new solution provides a better value in light of the objective function, the new solution replaces the current one. The search terminates if no better neighbor solution can be found.

One of the most popular local search methods for solving SAT is GSAT [9]. The GSAT algorithm operates by changing a complete assignment of variables into one in which the maximum possible number of clauses are satisfied by changing the value of a single variable. An extension of GSAT referred as random-walk [10] has been realized with the purpose of escaping from local optima. In a random walk step, an unsatisfied clause is randomly selected. Then, one of the variables appearing in that clause is flipped, thus effectively forcing the selected clause to become satisfied. The main idea is to decide at each search step whether to perform a standard GSAT or a random-walk strategy with a probability called the walk probability. Another widely used variant of GSAT is the WalkSAT algorithm originally introduced in [12]. It first picks randomly an unsatisfied clause and then in a second step, one of the variables with the lowest break count appearing in the selected clause is randomly selected. The break count of a variable is defined as the number of clauses that would be unsatisfied by flipping the chosen variable. If there exists a variable with break count equals to zero, this variable is flipped, otherwise the variable with minimal break count is selected with a certain probability (noise probability). The choice of unsatisfied clauses combined with the randomness in the selection of variables enable WalkSAT to avoid local minima and to better explore the search space.

Recently, new algorithms [12] [13] [14] [15] [16] have emerged using history-based variable selection strategy in order to avoid flipping the same variable. Apart from GSAT and its variants, several clause weighting based SLS algorithms [17] [18] have been proposed to solve SAT problems. The key idea is associate the clauses of the given CNF formula with weights. Although these clause weighting SLS algorithms differ in the manner how clause weights should be updated (probabilistic or deterministic) they all choose to increase the weights of all the unsatisfied clauses as soon as a local minimum is encountered. Clause weighting acts as a diversification mechanism rather than a way of escaping local minima. Finally, many other SLS algorithms have been applied to SAT. These include techniques such as Simulated Annealing [19], Evolutionary Algorithms [20], and Greedy Randomized Adaptive Search Procedures [21].

## 3. The GSAT greedy algorithm

This section is devoted to explaining the GSAT greedy algorithm and one of its variants before embedding it into the multilevel paradigm. Basically, the GSAT algorithm begins with a randomly generated assignment of the variables, and then uses the steepest descent

heuristic to find the new truth value assignment which best decreases the number of unsatisfied clauses. After a fixed number of moves, the search is restarted from a new random assignment. The search continues until a solution is found or a fixed number of restart is performed. As with any combinatorial optimization , local minima or plateaus (i.e., a set of neighboring states each with an equal number of unsatisfied clauses) in the search space are problematic in the application of greedy algorithms. A local minimum is defined as a state whose local neighborhood does not include a state that is strictly better. The introduction of an element of randomness (e.g., noise) into a local search methods is a common practice to increase the success of GSAT and improve its effectiveness through diversification [2].

```
Procedure GSAT Random Walk

Input: A set of clauses, MAX_TRIES, MAX_FLIPS, walking probability p;

Output: A satisfying truth assignment of the clauses, if found;

Begin
    /* Initialization */
    For i := 1 To MAX_TRIES Do
        T ← a random truth assignment;
        For j := 1 To MAX_FLIPS Do
            If T satisfies all the clauses Then return T;
            r ← a randomly generated number between 0 and 1;
            If r ≤ p Then
              PossFlips ← a randomly selected variable with the
              largest decrease in the number of unsatisfied clauses;
            Else
              PossFlips ← a random variable in some unsatisfied clause;
            EndIf
            v ← Pick (PossFlips);
            T ← T with v flipped;
        EndFor
    EndFor
    return "no solution found"
End
```

Fig. 1. The GSAT Random Walk Algorithm.

The algorithm of GSAT Random Walk, which is shown in Figure 1, starts with a randomly chosen assignment. Thereafter two possible criteria are used in order to select the variable to be flipped. The first criterion uses the notion of a "noise" or walk-step probability to randomly select a currently unsatisfied clause and flip one of the variables appearing in it also in a random manner. At each walk-step, at least one unsatisfied clause becomes satisfied. The other criterion uses a greedy search to choose a random variable from the set PossFlips. Each variable in this set, when flipped, can achieve the largest decrease (or the least increase) in the total number of unsatisfied clauses. The walk-step strategy may lead to an increase in the total number of unsatisfied clauses even if improving flips would have been possible. In consequence, the chances of escaping from local minima of the objective function are better compared with the basic GSAT [11].

## 4. The multilevel paradigm

The *multilevel* paradigm is a simple technique which at its core applies recursive coarsening to produce smaller and smaller problems that are easier to solve than the original one. Figure 2 shows the generic multilevel paradigm in pseudo-code. The multilevel paradigm consists of three phases: coarsening, initial solution, and multilevel refinement. During the coarsening phase, a series of smaller problems is constructed by matching pairs of vertices of the input original problem in order to form clusters, which define a related coarser problem. The coarsening procedure recursively iterates until a sufficiently small problem is obtained. Computation of an initial solution is performed on the coarsest level (the smallest problem). Finally, the initial solution is projected backward level by level. Each of the finer levels receives the preceding solution as its initial assignment and tries to refine it by some local search algorithm. A common feature that characterizes multilevel algorithms is that any solution in any of the coarsened problems is a legitimate solution to the original problem. This is always true as long as the coarsening is achieved in a way that each of the coarsened problems retains the original problem's global structure.

```
Procedure Multilevel Paradigm

Input: SAT problem P_0

Output: Solution S_final(P_0)

Begin
    level := 0
    /* Coarsening Phase */
    While (the desired number of levels is not reached)
            P_level+1 := Coarsen (P_level)
            level := level + 1
    end
    /* Initial Solution is computed at the lowest level */
    S(P_level) = Initial Solution (P_level)
    /* Uncoarsening and Refinement phase */
    While (level > 0)
            S_start(P_level-1) := Uncoarsen (S_final(P_level))
            S_final(P_level-1) := Refine (S_start(P_level-1))
            level := level - 1
    end
End
```

Fig. 2. The Multilevel Generic Algorithm.

The key success behind the efficiency of the multilevel techniques is the use of the multilevel paradigm, which offers two main advantages enabling local search techniques to become much more powerful in the multilevel context. First, by allowing local search schemes to view a cluster of vertices as a single entity, the search becomes restricted to only those configurations in the solution space in which the vertices grouped within a cluster are assigned the same label. During the refinement phase a local refinement scheme applies a local transformation within the neighborhood (i.e., the set of solutions that can be reached

from the current one) of the current solution to generate a new one. As the size of the clusters varies from one level to another, the size of the neighborhood becomes adaptive and allows the possibility of exploring different regions in the search space. Second, the ability of a refinement algorithm to manipulate clusters of vertices provides the search with an efficient mechanism to escape from local minima.

Multilevel techniques were first introduced when dealing with the graph partitioning problem (GPP) [1] [5] [6] [7] [8] [22] and have proved to be effective in producing high quality solutions at lower cost than single level techniques. The traveling salesman problem (TSP) was the second combinatorial optimization problem to which the multilevel paradigm was applied and has clearly shown a clear improvement in the asymptotic convergence of the solution quality. Finally, when the multilevel paradigm was applied to the graph coloring problem, the results do not seem to be in line with the general trend observed in GPP and TSP as its ability to enhance the convergence behaviour of the local search algorithms was rather restricted to some problem classes.

## 5. A multilevel framework for SAT

- **Coarsening:** The original problem $P_0$ is reduced into a sequence of smaller problems $P_0$, $P_2$, . . . , $P_m$. It will require at least $\mathcal{O}(\log n/n')$ steps to coarsen an original problem with $n$ variables down to $n'$ variables. Let $\mathcal{V}_i^v$ denote the set of variables of $P_i$ that are combined to form a single variable $v$ in the coarser problem $P_{i+1}$. We will refer to $v$ as a multivariable. Starting from the original problem $P_0$, the first coarser problem $P_1$ is constructed by matching pairs of variables of $P_0$ into multivariables. The variables in $P_0$ are visited in a random order. If a variable has not been matched yet, then we randomly select another unmatched variable, and a multivariable consisting of these two variables is created. Unmatchable variables are simply copied to the coarser level. The new multivariables are used to define a new and smaller problem. This coarsening process is recursively carried out until the size of the problem reaches some desired threshold.
- **Initial solution:** An initial assignment $\mathcal{A}_m$ of $P_m$ is easily computed using a random assignment algorithm, which works by randomly assigning to each multivariable of the coarsest problem $P_m$ the value of true or false.
- **Projection:** Having optimized the assignment $\mathcal{A}_{k+1}$ for $P_{k+1}$, the assignment must be projected back to its parent $P_k$. Since each multivariable of $P_{k+1}$ contains a distinct subset of multivariables of $P_k$, obtaining $\mathcal{A}_k$ from $\mathcal{A}_{k+1}$ is done by simply assigning the set of variables $\mathcal{V}_k^v$ the same value as $v \in P_{k+1}$ (i.e., $A_k[u] = A_{k+1}[v]$, $\forall u \in \mathcal{V}_k^v$).
- **Refinement:** At each level, the assignment from the previous coarser level is projected back to give an initial assignment and further refined. Although $\mathcal{A}_{k+1}$ is a local minimum of $P_{k+1}$, the projected assignment $\mathcal{A}_k$ may not be at a local optimum with respect to $P_k$. Since $P_k$ is finer, it may still be possible to improve the projected assignment using a version of GSAT adapted to the multilevel paradigm. The idea of GSAT refinement as shown in Figure 3 is to use the projected assignment of $P_{k+1}$ onto $P_k$ as the initial assignment of GSAT. Since the projected assignment is already a good one, GSAT will hopefully converge quickly to a better assignment. During each level, GSAT is allowed to perform MAXFLIPS iterations before moving to a finer level. If a solution is not

found at the finest level, a new round of coarsening, random initial assignment, and refinement is performed.

```
Procedure GSAT Refine

Input: P_i, A_i, MAX_FLIPS;

Output: A satisfying truth assignment of the clauses, if found;

Begin
    For j := 1 To MAX_FLIPS Do
        If A_i satisfies all the clauses of P_i Then return A_i;
        PossFlips ← a randomly selected multivariable with the
        largest decrease in the number of unsatisfied clauses;
        v ← Pick (PossFlips);
        A_i ← A_i with v flipped;
    EndFor
End
```

Fig. 3. The GSAT Refinement Algorithm.

## 6. Experimental results

### 6.1 Benchmark instances

To illustrate the potential gains offered by the multilevel greedy algorithm, we selected a benchmark suite from different domains including benchmark instances of SAT competition Beijing held in 1996. These instances are by no means intended to be exhaustive but rather as an indication of typical performance behavior. All these benchmark instances are known to be hard and difficult to solve and are available from the SATLIB website (http://www.informatik.tudarmstadt. de/AI/SATLIB). All the benchmark instances used in this section are satisfiable and have been used widely in the literature in order to give an overall picture of the performance of different algorithms. Due to the randomization of the algorithm, the time required for solving a problem instance varies between different runs. Therefore, for each problem instance, we run GSAT and MLVGSAT both 100 times with a max-time cutoff parameter set to 300 seconds. All the plots are given in logarithmic scale showing the evolution of the solution quality based on averaged results over the 100 runs.

#### 6.1.1 Random-3-SAT

Uniform Random-3-SAT is a family of SAT problems obtained by randomly generating 3-CNF formula in the following way: For an instance with $n$ variables and $k$ clauses, each of the $k$ clauses is constructed from 3 literals which are randomly drawn from the $2n$ possible literals (the $n$ variables and their negations), such that each possible literal is selected with the same probability of $1/2n$. Clauses are not accepted for the construction of the problem instance if they contain multiple copies of the same literal or if they are tautological (i.e., they contain a variable and its negation as a literal).

#### 6.1.2 SAT-encoded graph coloring problems

The graph coloring problem (GCP) is a well-known combinatorial problem from graph theory: Given a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = v_1, v_2, \ldots, v_n$ is the set of vertices and $\mathcal{E}$ the set of

edges connecting the vertices, the goal is to find a coloring $\mathcal{C} : \mathcal{V} \rightarrow N$, such that two vertices connected by an edge always have different colors. There are two variants of this problem: In the optimization variant, the goal is to find a coloring with a minimal number of colors, whereas in the decision variant, the question is to decide whether for a particular number of colours, a coloring of the given graph exists. In the context of SAT-encoded graph coloring problems, we focus on the decision variant.

#### 6.1.3 SAT-encoded logistics problems

In the logistics planning domain, packages have to be moved between different locations in different cities. Within cities, packages are carried by trucks while between cities they are transported by planes. Both trucks and airplanes are of limited capacity. The problem involves 3 operators (load, unload, move) and two state predicates (in, at). The initial and goal state specify locations for all packages, trucks, and planes; the plans allow multiple actions to be executed simultaneously, as long as no conflicts arise from their preconditions and effects. The question in the decision variant is to decide whether a plan of a given length exists. SAT-based approaches to logistics planning typically focus on the decision variant.

#### 6.1.4 SAT-encoded block world planning problems

The Blocks World is a very well-known problem domain in artificial intelligence research. The general scenario in Blocks World Planning comprises a number of blocks and a table. The blocks can be piled onto each other, where the down-most block of a pile is always on the table. There is only one operator which moves the top block of a pile to the top of another pile or onto the table. Given an initial and a goal configuration of blocks, the problem is to find a sequence of operators which, when applied to the initial configuration, leads to the goal situation. Such a sequence is called a (linear) plan. Blocks can only be moved when they are clear, i.e., no other block is piled on top of them, and they can be only moved on top of blocks which are clear or onto the table. If these conditions are satisfied, the move operator always succeeds. SAT-based approaches to Blocks World Planning typically focus on the decision variant where the question is to decide whether a plan of a given length exists.

#### 6.1.5 SAT-encoded quasigroup problems

A quasigroup is an ordered pair $(\mathcal{Q}, \cdot)$, where $\mathcal{Q}$ is a set and $\cdot$ is a binary operation on $\mathcal{Q}$ such that the equations $a \cdot x = b$ and $y \cdot a = b$ are uniquely solvable for every pair of elements $a, b$ in $\mathcal{Q}$. The cardinality of the set $\mathcal{Q}$ is called the order of the quasigroup. Let $N$ be the order of the quasigroup $\mathcal{Q}$ then the multiplication table $\mathcal{Q}$ is a table $N \times N$ such that the cell at the coordinate $(x, y)$ contains the result of the operation $x \cdot y$. The multiplication of the quasigroup must satisfy a condition that there are no repeated result in each row or column. Thus, the multiplication table defines a Latin square. A complete Latin square is a table that is filled completely. The problem of finding a complete Latin square can be stated as a satisfiability problem.

### 6.2 Experimental results

Figures 4-15 show individual results which appear to follow the same pattern within each application domain. Overall, at least for the instances tested here, we observe that the search

pattern happens in two phases. In the first phase, both MLVGSAT and GSAT behave as a hill-climbing method. This phase is short and a large number of the clauses are satisfied. The best assignment climbs rapidly at first, and then flattens off as we mount the plateau, marking the start of the second phase. The plateau spans a region in the search space where flips typically leave the best assignment unchanged. The long plateaus become even more pronounced as the number of flips increases, and occurs more specifically in trying to satisfy the last few remaining clauses. The transition between two plateaus corresponds to a change to the region where a small number of flips gradually improve the score of the current solution ending with an improvement of the best assignment. The plateau is rather of short length with MLVGSAT compared with that of GSAT. For MLVGSAT the projected solution from one level to its finer predecessor offers an elegant mechanism to reduce the length of the plateau as it consists of more degrees of freedom that can be used for further improving the best solution. The plots show a time overhead for MLVGSAT specially for large problems due mainly to data structures settings at each level. We feel that this initial overhead, which is a common feature in multilevel implementations is more susceptible to further improvements, and will be considerably minimized by a more efficient implementation. Comparing GSAT and MLVGSAT for small problems (up to 1500 clauses) and as can be seen from the left sides of Figures 6,8, both algorithms seem to be reaching the optimal quality solutions. It is not immediately clear which of the two algorithms converges more rapidly. This is probably very dependent on the choice of the instances in the test suite. For example the run time required by MLVGSAT for solving instance flat100-239 is more than 12 times higher than the mean run-time of GSAT (25sec vs 2sec). The situation is reversed when solving the instance block-medium (20sec vs 70sec). The difference in convergence behavior of the two algorithms becomes more distinctive as the size of the problem increases. All the plots show a clear dominance of MLGSAT over GSAT throughout the whole run. MLVGSAT shows a better asymptotic convergence (to around 0.008%−0.1%) in excess of the optimal solution as compared with GSAT which only reach around (0.01%- 11%). The performance of MLVGSAT surpasses that of GSAT although few of the curves overlay each other closely, MLVGSAT has marginally better asymptotic convergence.

The quality of the solution may vary significantly from run to run on the same problem instance due to random initial solutions and subsequent randomized decisions. We choose the Wilcoxon Rank test in order to test the level of statistical confidence in differences between the mean percentage excess deviation from the solution of the two algorithms. The test requires that the absolute values of the differences between the mean percentage excess deviation from the solution of the two algorithms are sorted from smallest to largest and these differences are ranked according to the absolute magnitude. The sum of the ranks is then formed for the negative and positive differences separately. As the size of the trials increases, the rank sum statistic becomes normal. If the null hypothesis is true, the sum of ranks of the positive differences should be about the same as the sum of the ranks of the negative differences. Using two-tailed $P$ value, significance performance difference is granted if the Wilcoxon test is significant for $P < 0.05$.

Looking at Table 1, we observe that the difference in the mean excess deviation from the solution is significant for large problems and remains insignificant for small problems.

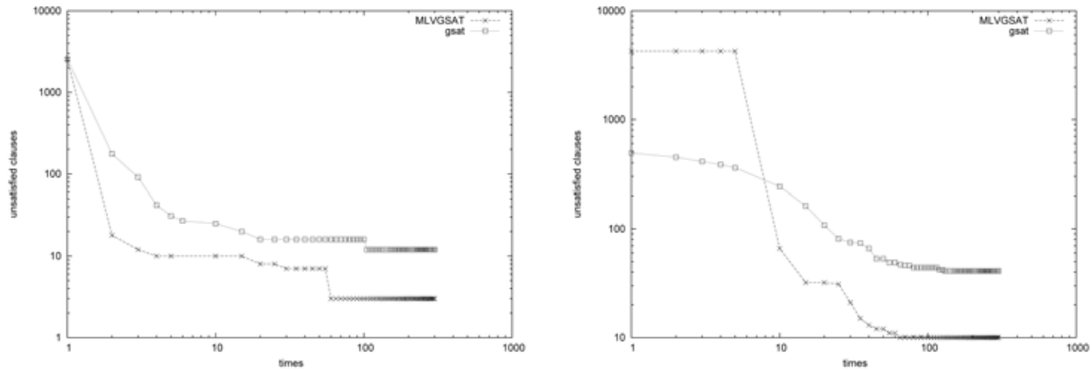



Fig. 4. Log-Log plot:Random:(Left ) Evolution of the best solution on a 600 variable problem with 2550 clauses (f600.cnf). Along the horizontal axis we give the time in seconds , and along the vertical axis the number of unsatisfied clauses. (Right) Evolution of the best solution on a 1000 variable problem with 4250 clauses. (f1000.cnf).Horizontal axis gives the time in seconds, and the vertical axis shows the number of unsatisfied clauses.

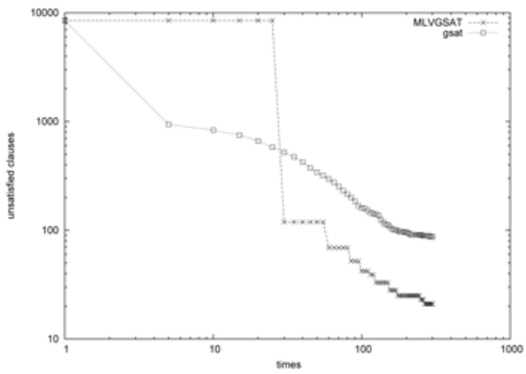



Fig. 5. Log-Log plot: Random:Evolution of the best solution on a 2000 variable problem with 8500 clauses (f2000.cnf). Along the horizontal axis we give the time in seconds , and along the vertical axis the number of unsatisfied clauses.

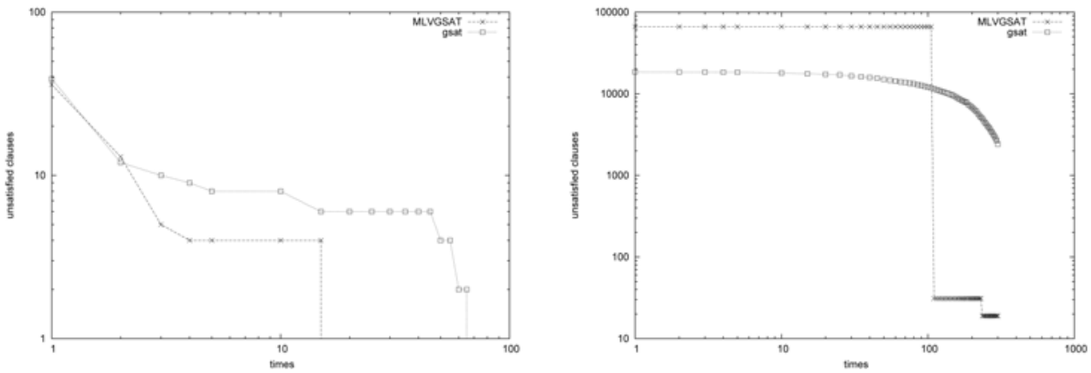



Fig. 6. Log-Log plot: SAT-encoded graph coloring:(Left ) Evolution of the best solution on a 300 variable problem with 1117 clauses (flat100.cnf). Along the horizontal axis we give the time in seconds, and along the vertical axis the number of unsatisfied clauses. (Right) Evolution of the best solution on a 2125 variable problem with 66272 clauses (g125-17.cnf). Horizontal axis gives the time in seconds, and the vertical axis shows the number of unsatisfied clauses.

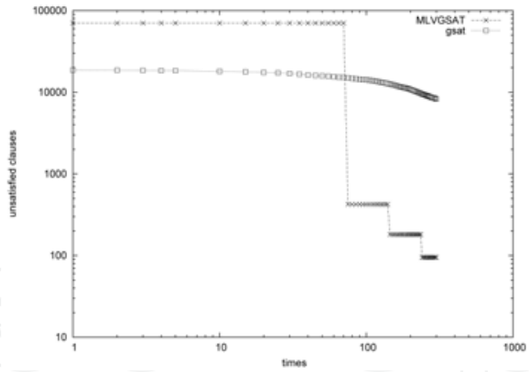



Fig. 7. Log-Log plot: SAT-encoded graph coloring:Evolution of the best solution on a 2250 variable problem with 70163 clauses (g125-18.cnf). Along the horizontal axis we give the time in seconds , and along the vertical axis the number of unsatisfied clauses.

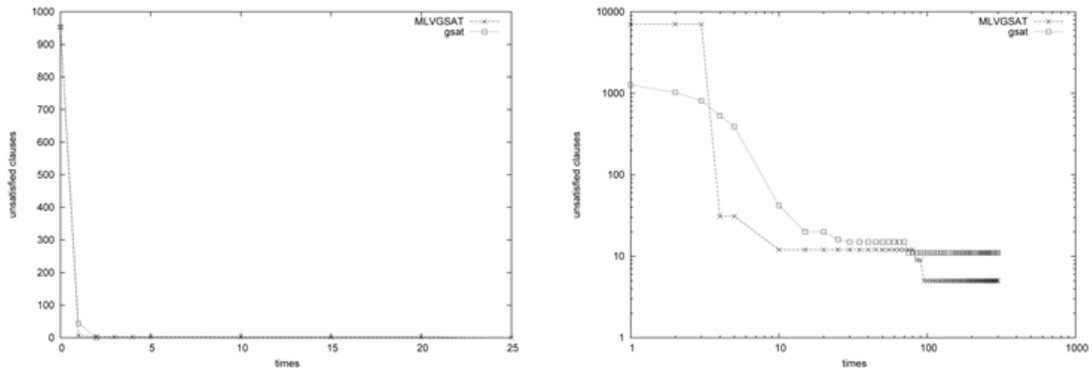



Fig. 8. SAT-encoded block world:(Left ) Evolution of the best solution on a 116 variable problem with 953 clauses (medium.cnf). Along the horizontal axis we give the time in seconds , and along the vertical axis the number of unsatisfied clauses. Log-Log plot (Right) Evolution of the best solution on a 459 variable problem with 7054 clauses (huge.cnf). Horizontal axis gives the time in seconds, and the vertical axis shows the number of unsatisfied clauses.

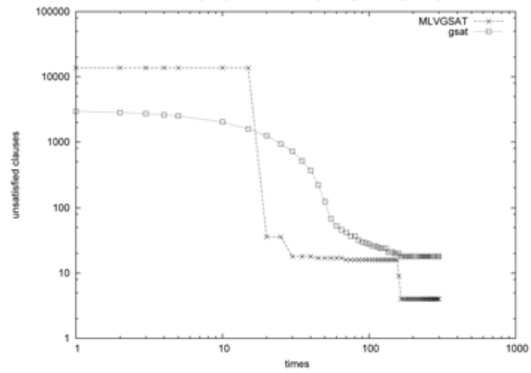



Fig. 9. Log-Log plot: SAT-encoded block world:Evolution of the best solution on a 1087 variable problem with 13772 clauses (bw-largeb.cnf). Along the horizontal axis we give the time in seconds, and along the vertical axis the number of unsatisfied clauses.

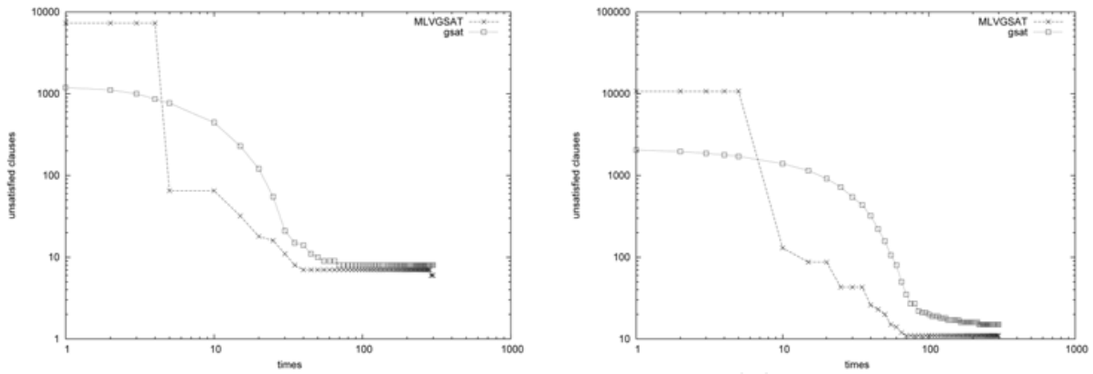



Fig. 10. Log-Log plot: SAT-encoded Logistics:(Left) Evolution of the best solution on a 843 variable problem with 7301 clauses (logisticsb.cnf). Along the horizontal axis is the time in seconds , and along the vertical axis the number of unsatisfied clauses. (Right) Evolution of the best solution on a 1141 variable problem with 10719 clauses (logisticsc.cnf). Horizontal axis gives the time in seconds, and the vertical axis shows the number of unsatisfied clauses.

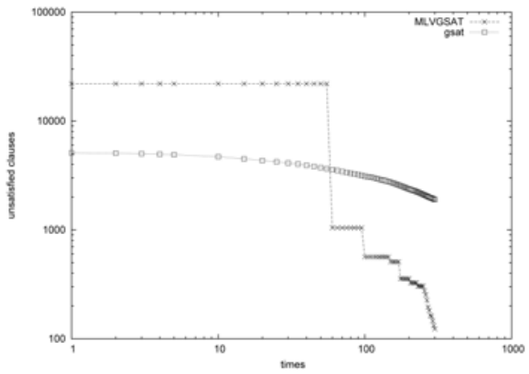



Fig. 11. Log-Log plot:SAT-encoded logistics:Evolution of the best solution on a 4713 variable problem with 21991 clauses (logisticsd.cnf). Along the horizontal axis we give the time in seconds, and along the vertical axis the number of unsatisfied clauses.

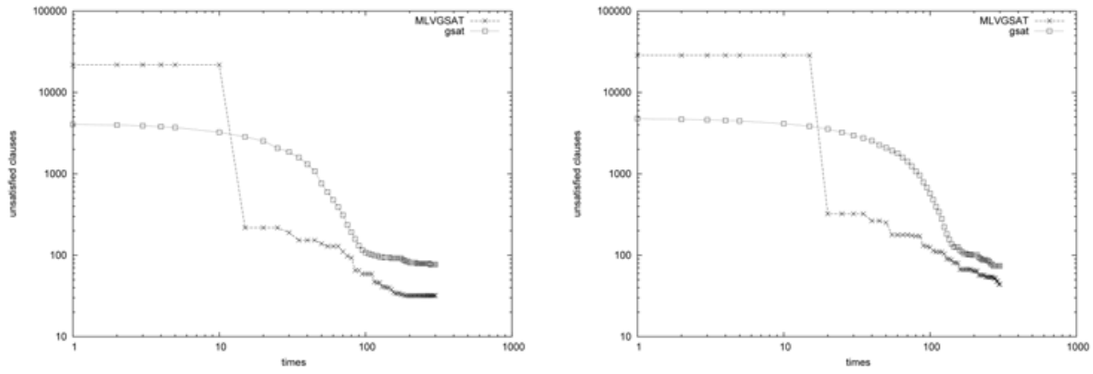



Fig. 12. Log-Log plot:SAT-encoded quasigroup:(Left) Evolution of the best solution on a 129 variable problem with 21844 clauses (qg6-9.cnf). Along the horizontal axis we give the time in seconds , and along the vertical axis the number of unsatisfied clauses.(Right) Evolution of the best solution on a 729 variable problem with 28540 clauses (qg5.cnf). Horizontal axis gives the time in seconds, and the vertical axis shows the number of unsatisfied clauses.

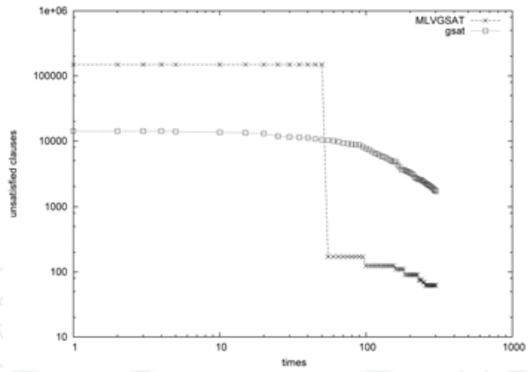



Fig. 13. Log-Log plot:SAT-encoded quasigroup:Evolution of the best solution on a 512 variable problem with 148957 clauses (qg1-8.cnf). Along the horizontal axis we give the time in seconds , and along the vertical axis the number of unsatisfied clauses.

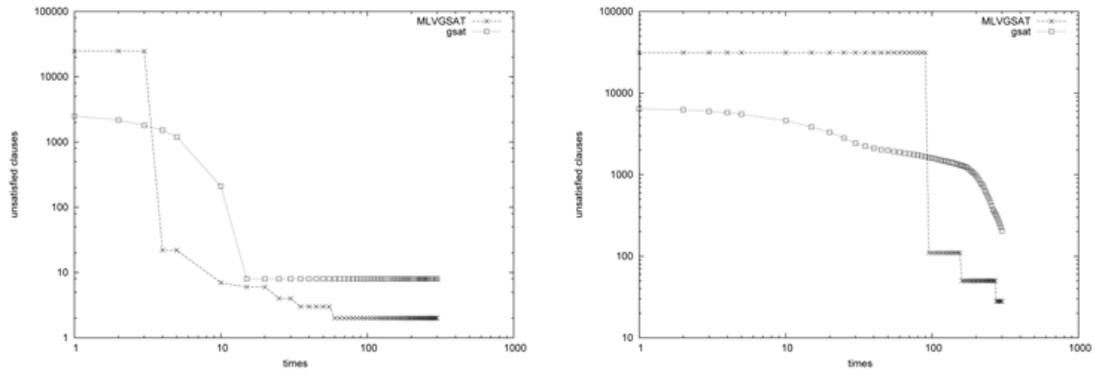



Fig. 14. Log-Log plot:SAT competition Beijing: (Left) Evolution of the best solution on a 410 variable problem with 24758 clauses (4blockb.cnf). Along the horizontal axis we give the time in seconds, and along the vertical axis the number of unsatisfied clauses. (Right) Evolution of the best solution on a 8432 variable problem with 31310 clauses (3bitadd-31.cnf). Horizontal axis gives the time in seconds, and the vertical axis shows the number of unsatisfied clauses.

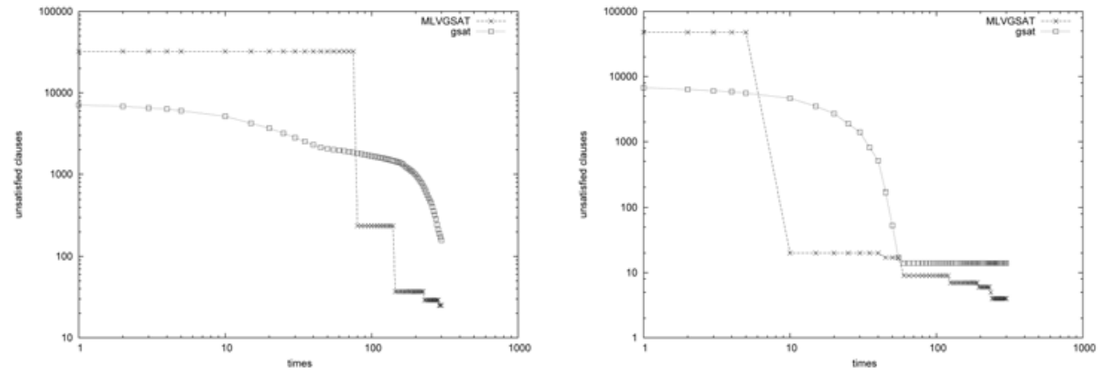



Fig. 15. Log-Log plot:SAT competition Beijing:(Left) Evolution of the best solution on a 8704 variable problem with 32316 clauses (3bitadd32.cnf). Along the horizontal axis we give the time in seconds, and along the vertical axis the number of unsatisfied clauses. (Right) Evolution of the best solution on a 758 variable problem with 47820 clauses (4blocks.cnf). Horizontal axis gives the time in seconds, and the vertical axis shows the number of unsatisfied clauses.

| Problem | %EX: MLVGSAT | %EX: GSAT | Null Hypotheis |
|---|---|---|---|
| f600 | 0.001 | 0.004 | Accept |
| f1000 | 0.002 | 0.009 | Accept |
| f2000 | 0.002 | 0.01 | Accept |
| flat100 | 0 | 0 | Reject |
| g125-17 | 0.0002 | 0.001 | Accept |
| g125-18 | 0.001 | 0.11 | Accept |
| logistic-b | 0.00008 | 0.0001 | Accept |
| logistic-c | 0.001 | 0.004 | Accept |
| logistic-d | 0.005 | 0.08 | Accept |
| bw-medium | 0 | 0 | Reject |
| bw-huge | 0.0007 | 0.001 | Accept |
| bw-large-b | 0.0002 | 0.001 | Accept |
| qg6-9 | 0.001 | 0.003 | Accept |
| qg5-9 | 0.001 | 0.002 | Accept |
| qg1-8 | 0.0004 | 0.011 | Accept |
| 4block | 0.00008 | 0.0008 | Accept |
| 3bitadd-31 | 0.0008 | 0.006 | Accept |
| 3bitadd-32 | 0.0007 | 0.004 | Accept |
| 4blocksb | 0.00008 | 0.0002 | Accept |

Table 1. Wilcoxon statistical test.

## 7. Conclusions

In this chapter, we have described and tested a new approach to solving the SAT problem based on combining the multilevel paradigm with the GSAT greedy algorithm. The resulting MLVGSAT algorithm progressively coarsens the problem, provides an initial assignment at the coarsest level, and then iteratively refines it backward level by level. In order to get a comprehensive picture of the new algorithm's performance, we used a benchmark set consisting of SAT-encoded problems from various domains. Based on the analysis of the results, we observed that within the same computational time, MLVGSAT provides higher quality solution compared with that of GSAT. Other conclusions that we may draw from the results are that the multilevel paradigm can either speed up GSAT or even improve its asymptotic convergence. Results indicated that the larger the instance, the higher the difference between the mean percentage excess deviation from the solution. An obvious subject for further work would be the use of efficient data structures in order to minimize the overhead during the coarsening and refinement phases. It would be of great interest to further validate or contradict the conclusions of this work by extending the range of problem classes. Finally, obvious subjects for further work include designing different coarsening strategies and tuning the refinement process.

## 8. References

[1] S.T. Barnard and H.D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101-17, 1994.

[2] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268-308, 2003.

[3] S.A. Cook. The complexity of theorem-proving procedures. *Proceedings of the Third ACM Symposium on Theory of Computing*, pages 151-158, 1971.

[4] M. Davis and H.Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201-215, 1960.

[5] R. Hadany and D. Harel. A Multilevel-Scale Algorithm for Drawing Graphs Nicely. *Tech.Rep.CS99-01*, Weizmann Inst.Sci, Faculty Maths.Comp.Sci, 1999.

[6] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In S. Karin, editor, *Proc.Supercomputing'95, San Diego*, 1995. ACM Press, New York.

[7] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J.Sci. Comput.*, 20(1):359-392, 1998.

[8] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *J.Par.Dist.Comput.*, 48(1):96-129, 1998.

[9] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. *Proceedings of AAA'92*, pages 440-446. MIT Press, 1992.

[10] B. Selman, Henry A. Kautz, and B. Cohen. Noise strategies for improving local search. *Proceedings of AAAI'94*, pages 337-343. MIT Press, 1994.

[11] B. Selman and H.K. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In R. Bajcsy, editor, *Proceedings of the international Joint Conference on Artificial Intelligence*, volume 1, pages 290-295. Morgan Kaufmann Publishers Inc., 1993.

[12] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. *Proceedings of AAAI'97*, pages 321-326. MIT Press, 1997.

[13] F. Glover. Tabu search – Part I. *ORSA Journal on Computing*, 1(3):190-206, 1989.

[14] P. Hansen and B. Jaumand. Algorithms for the maximum satisfiability problem. *Computing*, 44:279-303, 1990.

[15] I. Gent and T. Walsh. Unsatisfied variables in local search. In J. Hallam, editor, *Hybrid Problems, Hybrid Solutions*, pages 73-85. IOS Press, 1995.

[16] L.P. Gent and T.Walsh. Towards an understanding of hill-climbing procedures for SAT. *Proceedings of AAAI'93*, pages 28-33. MIT Press, 1993.

[17] B. Cha and K. Iwama. Performance tests of local search algorithms using new types of random CNF formula. *Proceedings of IJCAI'95*, pages 304-309. Morgan Kaufmann Publishers, 1995.

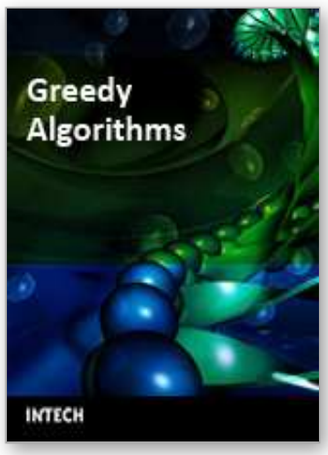
[18] J. Frank. Learning short-term clause weights for GSAT. *Proceedings of IJCAI'97*, pages 384- 389. Morgan Kaufmann Publishers, 1997.

[19] W.M. Spears. Simulated Annealing for Hard Satisfiability Problems. *Technical Report*, Naval Research Laboratory, Washington D.C., 1993.

[20] A.E. Eiben and J.K. van der Hauw. Solving 3-SAT with adaptive genetic algorithms. *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pages 81-86. IEEE Press, 1997.

[21] D.S. Johnson and M.A. Trick, editors. Cliques, Coloring, and Satisfiability, *Volume 26 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[22] C. Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM J.Sci. Comput.*, 22(1):63-80, 2000.




**Greedy Algorithms**

Edited by Witold Bednorz

ISBN 978-953-7619-27-5
Hard cover, 586 pages
**Publisher** InTech
**Published online** 01, November, 2008
**Published in print edition** November, 2008

Each chapter comprises a separate study on some optimization problem giving both an introductory look into the theory the problem comes from and some new developments invented by author(s). Usually some elementary knowledge is assumed, yet all the required facts are quoted mostly in examples, remarks or theorems.

**How to reference**

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Noureddine Bouhmala and Xing Cai (2008). A Multilevel Greedy Algorithm for the Satisfiability Problem, Greedy Algorithms, Witold Bednorz (Ed.), ISBN: 978-953-7619-27-5, InTech, Available from: http://www.intechopen.com/books/greedy_algorithms/a_multilevel_greedy_algorithm_for_the_satisfiability_problem

INTECH
open science | open minds

**InTech Europe**
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen